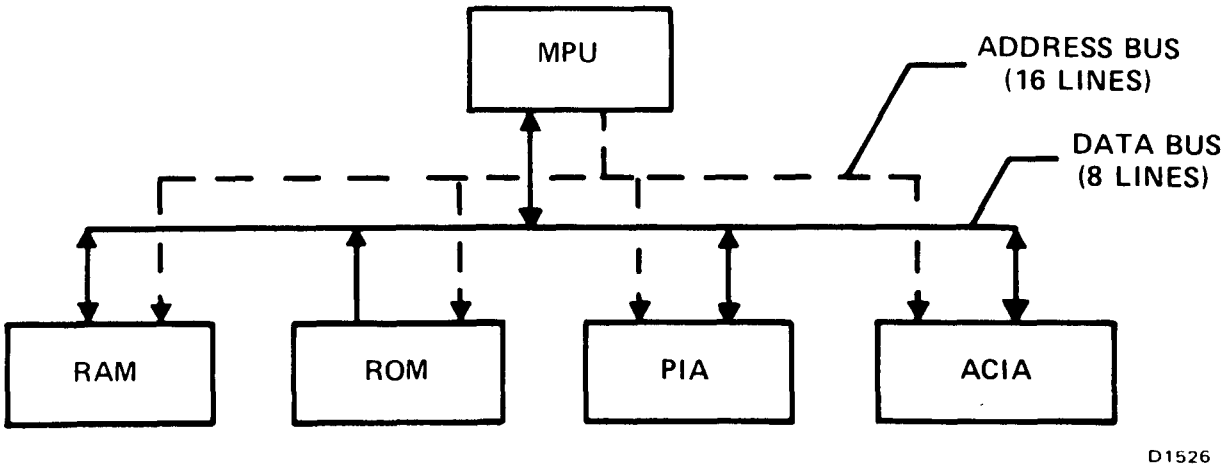
INTRODUCTION

The Motorola M6800 Microcomputer System of standard LSI (Large Scale Integration) devices permits the systems designer to configure and connect a total system with a minimum amount of time and effort. The MC6800 Microprocessing Unit (MPU) forms the nucleus of the system. LSI modules available which may be used to configure a total system in conjunction with the MC6800 MPU, include: 1) MC6810 Random Access Memory (RAM); 2) MC6830 Read Only Memory (ROM); 3) MC6820 Peripheral Interface Adapter (PIA), and 4) MC6850 Asynchronous Communications Interface Adapter (ACIA).

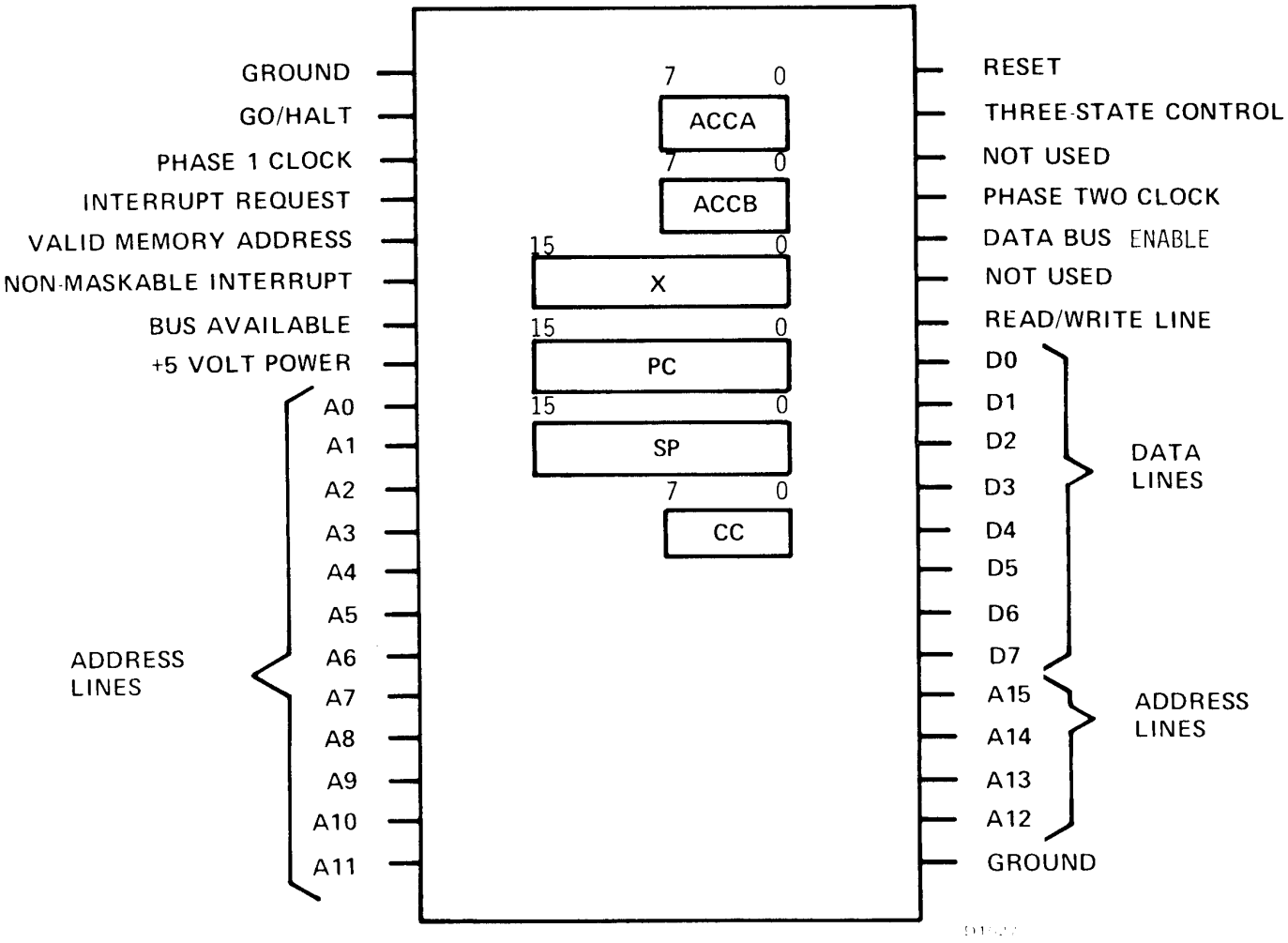
The MPU communicates with the rest of the system via a 16 bit address bus and an 8 bit data bus. The 16 bit address bus provides the MPU the capability of addressing up to 64K. The 8 bit data bus is bi-directional in that data is transferred both into the MPU or out of the MPU over the same bus. A read/write (R/W) line is provided to allow the MPU to control the direction of data transfer.. Since the same bus is used for both data into the MPU or out of the MPU, a separate 8 line bus is saved.

Other features of the M6800 system include a single +5 volt supply, operation at clock rates from 100 kilohertz to 1 megahertz, plus hardware and software interrupt capability.



Microprocessing Unit (MC6800)

The nucleus of the M6800 Microcomputer Family is the microprocessing unit (MPU). The MPU is enclosed in a 40 pin package as shown below:



Features included in the MPU are:

1. Two accumulators (ACCA and ACCB)
2. One index register (X)
3. One program counter register (PC)
4. One stack pointer register (SP)

5. One condition code register (CC)
6. 72 instructions
7. Five addressing modes
8. System clock range of 100 kilohertz to 1 megahertz
9. Program interrupt capability

Accumulators

The MPU contains 2 accumulators designated ACCA and ACCB. Each accumulator is 8 bits (one byte) long and is used to hold operands and data from the arithmetic logic unit. Instructions which involve one or both accumulators are:

ABA - Add accumulator A to accumulator B

ADC - Add with carry

ADD - Add without carry

AND - Logical AND

ASL - Arithmetic shift left

ASR - Arithmetic shift right

BIT - Bit test

CBA - Compare accumulators

CLR - Clear

CMP - Compare

COM - Complement

DAA - Decimal adjust ACCA

DEC - Decrement

EOR - Exclusive OR

INC – Increment

LDA - Load accumulator

LSR - Logical shift right

NEA - Negate

ORA - Inclusive OR

PSH - Push data onto stack

PUL - Pull data from stack

ROL - Rotate left

ROR - Rotate right

RTI - Return from interrupt

SBA - Subtract accumulators

SBC - Subtract with carry

STA - Store accumulator

SUB - Subtract

SWI - Software interrupt

TAB - Transfer from accumulator A to accumulator B

TAP - Transfer from accumulator A to processor condition codes register

TBA - Transfer from accumulator B to accumulator A

TPA - Transfer from processor condition codes register to accumulator A

TST - Test

WAI - Wait for interrupt

Index Register

The index register (X) is a 16 bit (2 byte) register which is primarily used to store a memory address in the Indexed mode of memory addressing. The index register may be decremented, incremented and stored. Instructions which involve the index register are:

CPX - Compare index register

DEX - Decrement index register

INX - Increment index register

LDX - Load index register

RTI - Return from interrupt

STX - Store index register

SWI - Software interrupt

TSX - Transfer stack pointer to index register

TXS - Transfer index register to stack pointer

WAI - Wait for interrupt

Program Counter

The program counter (PC) is a 16 bit register that contains the address of the next byte to be fetched from memory. When the current value of the program counter is placed on the address buss, the program counter will be incremented automatically.

Stack Pointer

The Stack Pointer (SP) is a 16 bit (2 byte) register that contains a beginning address, normally in RAM, where the status of the MPU registers may be stored when the MPU has other functions to perform, such as during an interrupt or during a Branch to Subroutine (BTS). The address in the stack pointer is the starting address of sequential memory locations in RAM where MPU status registers will be stored. The status of the MPU will be stored in the RAM as follows:

Stack Pointer Address : contents of PCL

Stack Pointer Address-1 : contents of PCH

Stack Pointer Address-2 : contents of IXL

Stack Pointer Address-3 : contents of IXH

Stack Pointer Address-4 : contents of ACCA

Stack Pointer Address-5 : Contents of ACCB

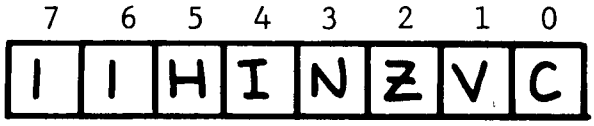
Stack Pointer Address-6 : Contents of CC

After the status of each register is stored on the stack, the Stack Pointer will be decremented. When the stack is unloaded (status of registers restored), the status of the last register on the stack will be the first register that is restored.

Condition Code Register (CC)

The condition code register is an 8 bit register. Each individual bit may get set or get cleared from execution of an instruction. To see how each instruction effects the condition code register, refer to the M6800 programming manual. The primary use of this register is execution of the conditional branch instruction. Bit 6 and 7 are not used and remain at logic "1."

CONDITION CODE REGISTER



BIT N0.	FUNCTION
0	C (Carry-Borrow Test)
1	V (Overflow Test)
2	Z (Zero Test)
3	N (Negative Test)
4	I (Interrupt Mask Test)
5	H (Half Carry Test)

Carry-Borrow: For addition, the carry-borrow condition code (C) in the zero bit position, represents a carry. This bit gets set (C=1) to indicate a carry, and is reset (C=0) if there is no carry.

For subtraction, the C bit is set (C=1) to indicate a borrow and is reset (C=0) to indicate there was no borrow.

Overflow: The V bit (bit 1) of the condition code register is set (V=1) when two's complement overflow results from an arithmetic operation, and is reset (V=O) if two's complement overflow does not occur.

Zero: The Z bit (bit 2) of the condition code register is set (Z=1) if the result of an arithmetic operation is zero, and is reset (Z=0) if the result is not zero.

Negative: The N bit (bit 3) of the condition code register is set (N=1) if bit 7 of an arithmetic operation is set (equal to 1). This indicates that the two's complement number, represented by the bit pattern of the result, is negative. The N bit is reset (N=0) if bit 7 of the arithmetic result is equal to 0.

Interrupt Mask: If this I bit (bit 4) is set (I=1), the MPU cannot respond to an interrupt request from any peripheral device.

Half-Carry: The half carry bit H (bit 5) of the condition code register is set (H=1) during execution of any of the instructions ABA,ADC, or ADD, if there is a carry from bit position 3 to bit position 4. The half carry is reset (H=0) during these operations, if there is no carry from bit position 4.

MPU Signal Descriptions

1. READ/WRITE (R/W):

This output line is used to signal all devices external to the MPU that the MPU is in a read state (R/W = High) or a write state (R/W = Low). The normal standby state of this line when no external devices are being accessed is a high state. This line is three-state. When three-state goes high, this line enters the high impedance mode.

2. VALID MEMORY ADDRESS(VMA):

This output line, (when in the high state) tells all devices external to the MPU that there is a valid address in the address bus. For RAM's and ROM's, this line should be ANDed with 12 clock and used as one of the enables. For PIA's, this line should be ANDed with one of the PIA address lines. This signal is not three-state.

3. DATA BUS ENABLE(DBE):

This signal will enable the data bus drives when in the high state. This input is normally the phase 2 (12) clock. During the high state, it will permit data to be output during a write cycle. During an MPU read cycle, the data bus drives will be disabled internally.

4. INTERRUPT REQUEST(IRQ):

This input from the PIA's requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the Condition Code Register is not set (interrupt masked), the machine will begin an interrupt sequence. The Index Register, Program

Counter, Accumulators, and Condition Code Register are stored away on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations n-6 and n-7 where n is the highest ROM address. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory.

5. Phase One (Øl)& Phase (Ø2)Clocks:

These two pins are used or a two phase non-overlapping clock that runs at the V DD voltage level.

These clocks run at a rate up to 1 megahertz.

6. Restart (RES):

RESTART (RES)--This input is used to start the MPU from a power down condition, resulting from a power failure or an initial start-up of the processor. If a positive edge is detected on the input, this will signal the MPU to begin the restart sequence. This will restart the MPU and start execution of a routine to initialize the processor. All the higher order address lines will be forced high. For the restart, the last 2 memory locations in the last ROM (n&n-1) will be accessed, whereby an address is stored which is the address to be loaded in the program counter which tells the processor where program execution is to begin.

7. NON-MASKABLE INTERRUPT(NMI):

This input requests that a nonmask-interrupt sequence be generated within the processor. As with the Interrupt

Request signal, the processor will complete the current instruction that is being executed before it recognizes the NMI signal. The interrupt mask bit in the Condition Code Register has no effect on NMI. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away on the stack. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations n-2 and n-3. An address loaded at these locations causes the MPU to branch to an nonmaskable interrupt routine in memory.

8. Go/Halt(G/H):

When this input is in the high state, the machine will fetch the instruction addressed by the program counter and start execution. When low all activity in the machine will be halted. This input is level sensitive. In the halt mode, the machine will stop at the end of an instruction. Bus Available will be at a logic "1" level. Valid Memory Address will be at a logic "0" and all other three-state lines will be in the three-state mode.

The halt line must go low with the leading edge of phase one to insure single instruction operation. If the halt line does not go low with the leading edge of phase one, one or two instruction operations may result, depending on when the halt line goes low relative to the phasing of the clock.

9. BUS AVAILABLE (BA):

The Bus Available signal will normally be in the low state. When activated, it will go to the high state indicating that the MPU has stopped and that the address bus is

available. This will occur if the GO/HALT line is in the Halt (low) mode or the 14PU is in a "Wait" state as the result of some instruction, such as the WAI instruction.

10. THREE-STATE CONTROL: (TSC)

This input causes all of the address lines and the Read/Write line to go into the off or high impedance state. The Valid Memory address and Bus Available signals will be forced low. The data bus is not affected by TSC and has its own enable (Data Bus Enable). In DMA applications, the Three-State Control line should be brought high on the leading edge of the Phase One Clock. The 11 clock must be held in the high state for this function to operate properly. The address bus will then be available for other devices to directly address memory. Since the MPU is a dynamic device, it must be refreshed periodically or destruction of data will occur.

11. ADDRESS BUS (AO/A15):

Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 130pf at 1 Megahertz.

When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications.

12. DATA BUS (DO/D7):

Eight pins are used for the data bus. It is bi-directional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load and 130pf at 1 Megahertz.

Microprocessor Instruction Set -- Alphabetic Sequence

ABA	Add Accumulators	INS	Increment Stack Pointer
ADC	Add with Carry	INX	Increment Index Register
ADD	Add		
AND	Logical And	JMP	Jump
ASL	Arithmetic Shift Left	JSR	Jump to Subroutine
ASR	Arithmetic Shift Right	LDA	Load Accumulator
		LDS	Load Stack Pointer
BCC	Branch if Carry Clear	LDX	Load Index Register
BCS	Branch if Carry Set	LSR	Logical Shift Right
BEQ	Branch if Equal to Zero		
BGE	Branch if Greater or Equal Zero	NEG	Negate
BGT	Branch if Greater than Zero	NOP	No Operation
BHI	Branch if Higher		
BIT	Bit Test	ORA	Inclusive OR Accumulator
BLE	Branch if Less or Equal	PSH	Push Data
BLS	Branch if Lower of Same	PUL	Pull Data
BLT	Branch if Less than Zero	ROL	Rotate Left
BMI	Branch if Minus	ROR	Rotate Right
BNE	Branch if Not Equal to Zero	RTI	Return from Interrupt
BPL	Branch if Plus	RTS	Return from Subroutine
BRA	Branch Always		
BSR	Branch to Subroutine	SBA	Subtract Accumulators
BVC	Branch if Overflow Clear	SBC	Subtract with Carry
BVS	Branch if Overflow Set	SEC	Set Carry
		SEI	Set Interrupt Mask
CBA	Compare Accumulators	SEV	Set Overflow
CLC	Clear Carry	STA	Store Accumulator
CLI	Clear Interrupt Mask	STS	Store Stack Register
CLR	Clear	STX	Store Index Register
CLV	Clear Overflow	SUB	Subtract
CMP	Compare	SWI	Software Interrupt
COM	Complement		
CPX	Compare Index Register	TAB	Transfer Accumulators
		TAP	Transfer Accumulators to Condition Code Reg.
DAA	Decimal Adjust	TBA	Transfer Accumulators
DEC	Decrement	TPA	Transfer Condition Code Reg. to Accumulator
DES	Decrement Stack Pointer	TST	Test
DEX	Decrement Index Register	TSX	Transfer Stack Pointer to Index Register
FOR	Exclusive OR	TXS	Transfer Index Register to Stack Pointer
INC	Increment	WAI	Wait for Interrupt

Hardware Interrupts

What happens when the MPU gets a hardware interrupt? After it has been determined that the interrupt is not non-maskable, the MPU checks the status of the mask bit (bit 4 of the condition code register). If the mask bit is set, the main program continues until a CLI (clears bit 4 of condition code register) instruction is executed, after which time the MPU will honor an interrupt by going to the stack pointer (SP) register and fetch an address which will be the 1st address in RAM where the status of the MPU registers will be stored during servicing of the interrupt.

```
SP      : contents of program counter low
SP-1    : contents of program counter high
SP-2    : contents of index register low
SP-3    : contents of index register high
SP-4    : contents of accumulator A
SP-5    : contents of accumulator B
SP-6    : contents of condition code register
```

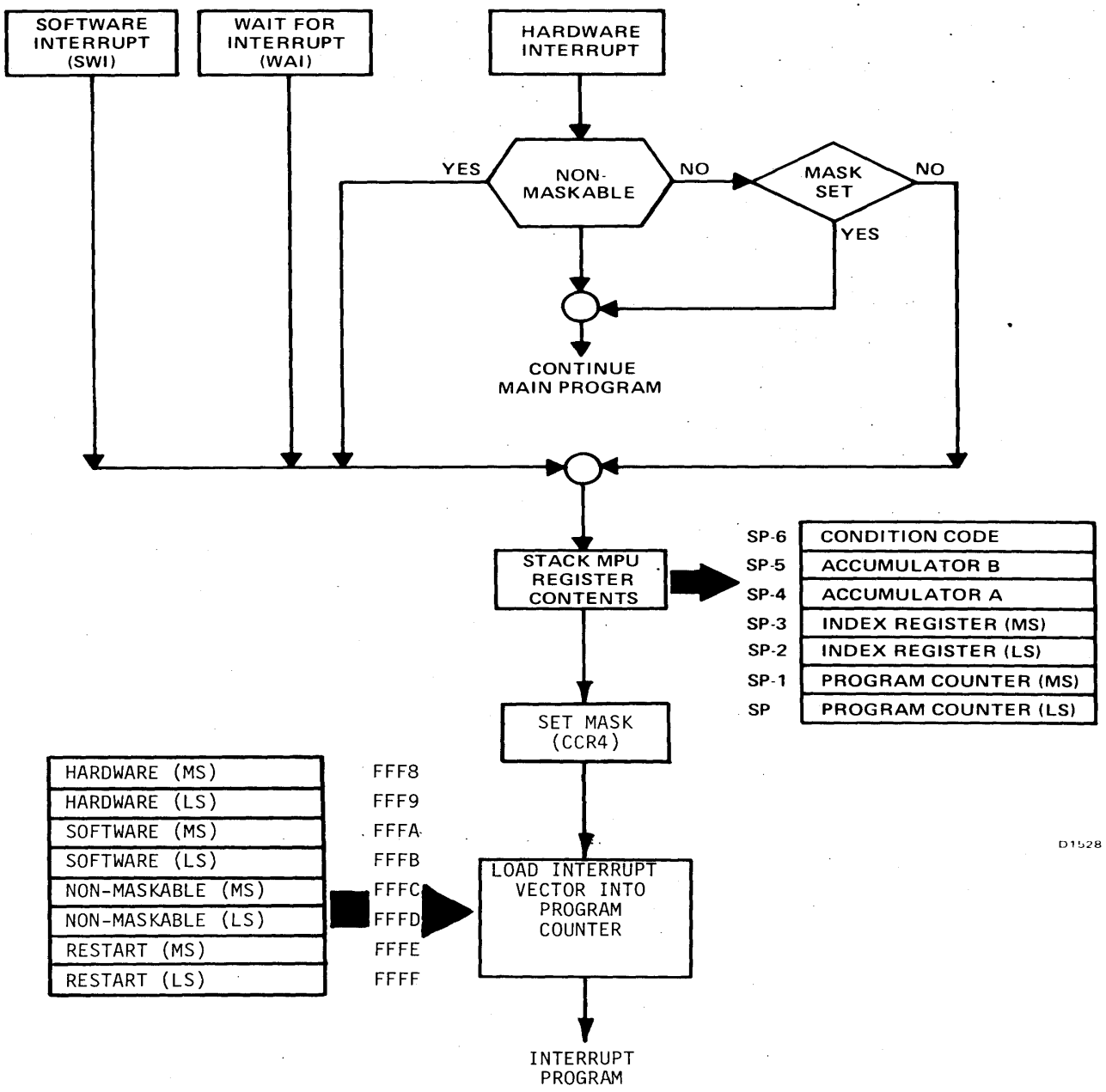
The address in the stack pointer register is determined by the programmer.

After the contents of the MPU registers have been stored in the stack, the mask bit is set thus preventing any further interrupts from interfering with the MPU until the program executes a CLI instruction. Next the MPU hardware automatically looks at addresses FFF8(MS) & FFF9 (LS) for the address of the poling routine to find out where the interrupt came from and what action to take.

After the interrupt has been serviced and an RTI instruction is executed, the stack, which contains the status of the registers before the interrupt, is unloaded in reverse order, i.e. the condition code register is loaded first, then accumulator B is restored, etc. When the registers have been restored to their status before the interrupt, the processor continues as though nothing happened.

The total story of interrupts is shown on the next two pages in the form of flow charts.

INTERRUPT FLOW CHART



SUMMARY OF MPU OPERATION

The MPU requires a two phase symmetrical, TTL compatible, nonoverlapping clock. During the first phase of the clock ($Ø_1$ high) an address will be placed on the address bus by the MPU. During the second phase of the clock ($Ø_2$ high), the bidirectional data bus will be active. The first byte of an instruction enters the MPU and is transferred into an internal instruction register and decoded by the MPU. The MPU will then contain the information needed to read in an additional one or two bytes of program is necessary. Once the entire instruction is read into the MPU (one, two or three bytes) the instruction is then executed. The MPU then reads in the next sequential byte in the program and places it again in the instruction register. The program will sequentially be executed in this manner unless a branch or jump instruction changes the value of the program counter. If this occurs, the next instruction to be executed is determined by the new program counter value.

If an interrupt or reset occurs during this process, the program counter value will also be changed. The new program counter value is determined by the highest eight memory locations that are reserved for reset and interrupt vectors.

In the case of interrupt, the stack pointer is used to store the contents of the internal registers necessary to return to the program location prior to the interrupt. This happens when the interrupt program exits by an RTI (Return from interrupt instruction). Similarly, the stack pointer is used to store the program counter value when a JSR (Jump to Subroutine) or BSR (Branch to Subroutine) instruction occurs. The program counter returns to its original value when an RTS (Return from Subroutine) instruction occurs. The stack pointer value is set by an LDS (Load Stack Pointer) instruction.

RESET SEQUENCE

1. While HALT is high, RESET goes low for at least eight cycles of $Ø_1$, $Ø_2$ during which all internal registers are cleared and interrupt bit (I) in CC is set.
2. Data at FFFE loads into PCH.
3. Data at FFFF loads into PCL.
4. PC contents go out on ADRS bus during $Ø_1$.
5. Contents of cell addressed enters instruction register during and is decoded as first instruction.
6. If two or more byte instruction, additional bytes enter MPU for execution. If not, go to next step.
7. After execution, step 5 is repeated for subsequent instructions.

IRQ SEQUENCE

1. If bit "I" in condition code register is not set (I = 0) and IRQ goes low for at least one $Ø_2$ cycle, the IRQ sequence will be entered.
2. After completion of the current instruction, internal registers PC, X, A, B and CC will be stored in RAM at the address indicated by the stack pointer in descending locations (7 bytes in all).
3. The IRQ mask (bit I = 1) is set.
4. Data at FFF8 gets loaded into PCH.
5. Data at FFF9 gets loaded into PCL.
6. PC contents go out on address bus during
7. Contents of call addressed enters instruction register during $Ø_2$ and is decoded as first instruction of interrupt routine.
8. If it is a more than 1 byte instruction, additional bytes enter MPU for execution. If not, go to next step.
9. After execution, step 5 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

NMI SEQUENCE

1. If NMI goes low for at least one $Ø_2$ cycle, the MPU will wait for completion of current instruction.
2. The internal registers PC, X, A, B and CC will then be stored in RAM at the address indicated by the stack pointer in descending locations (7 bytes in all).
3. The IRQ (bit I = 1) mask is set.
4. Data at FFFC is loaded into PCH.
5. Data at FFFD is loaded into PCL.
6. PC contents go out on ADRS bus during $Ø_1$.
7. Contents of cell addressed enters instruction register during $Ø_2$ and is decoded as first instruction of NMI subroutine.
8. If two or more byte instruction, additional bytes enter MPU for execution. If not, go to next step.
9. After execution, Step 5 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

RTI EXECUTION

1. The contents of the stack are loaded back into the MPU. (unwinds)
2. The contents of the PC go out on the address bus to fetch the first byte of the next instruction.

SWI INSTRUCTION

1. Contents of the MPU registers PC, 1X, ACCA, ACCB and CC are stored in RAM at the address indicated by the stack pointer in descending location (7 bytes in all).
2. The IRQ mask (bit I = 1) is set.
3. Data at FFFA gets loaded into PCH.
4. Data at FFFB gets loaded into PCL.
5. PC contents go out on address bus during $Ø_1$.
6. Contents of cell addressed enters instruction register during $Ø_2$ and is decoded as first instruction of SWI subroutine.
7. If it is a more than one byte instruction, additional bytes enter MPU for execution. If not, go to next step.
8. After execution, Step 6 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

Number Systems

Everyone is quite familiar with the base 10 number system i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, & 9, since this is the system we all use day to day. Let us review a typical number, say 2743, and see what it really means. The least significant digit (LSD) is 3 and the most significant digit (MSD) is 2. Since we are talking about a base 10 number, the number 2743 really is

$= 3x10^0 + 4x10^1 + 7x10^2 + 2x10^3$

$= 3x1 + 4x10 + 7x100 + 2x1000$

$= 3 + 40 + 700 + 2000$

$= 2743.$

In digital computers, base 10 numbers are represented in binary form, i.e. 1's & 0's. Lets take a base 10 number and convert it to a binary (base 2) number. A method of doing this is known as "repeated division by ?". The base 10 number of 47 is converted to binary as shown below:

```
   23
2 | 47    R=1

   11
2 | 23    R=1

    5
2 | 11    R=1

   2
2 | 5     R=1          101111₂

   2
2 | 2     R=0

   0
2 | 1     R=1
```

Converting 101111_2 back to our base 10 number is done in the same manner as above

$$101111_2 = 1x2^0 + 1x2^1 + 1x2^2 + 1x2^3 + 0x2^4 + 1x2^5$$
$$= 1x1 + 1x2 + 1x4 + 1x8 + 0x16 + 1x32$$
$$= 1 + 2 + 4 + 8 + 0 + 32$$
$$= 47_{10}$$

In general, converting from a number in any base to a number in base 10 is accomplished as follows:

$$(A_0\ B^0 + A_1\ B^1 + A_2\ B^2 + A_3\ B^3 + A_4\ B^4 \text{ -- -- -- -- } A_n\ B^n)$$

where B is the base of the number system and A is the particular digit in the original number corresponding to its position to the left of the decimal point. On the example just completed, (101111). $A_0 = 1$, $A_1 = 1$, $A_2 = 1$, $A_3 = 1$, $A_4 = 0$, & $A_5 = 1$ and B = 2 (base 2).

Another base which is very convenient in digital computers is base 8, since base 8 is really a convenient way of representing base 2. Lets illustrate by converting a base 10 number to base 8 & base 2. Let's convert 61 in base 10 to a number in base 8 and a number in base 2. By continuous division:

```
   7
8 | 61    R=5

   0
8 | 7     R=7          75(8)
```

```
   30
2 | 61    R=1

   15
2 | 30    R=0

    7
2 | 15    R=1

   3
2 | 7     R=1          111101(2)

   1
2 | 3     R=1

   0
2 | 1     R=1
```

First lets prove that 75_8 & 111101_2 are really equal to 61_{10}.

$$75_8 = 5x8^0 + 7x8^1$$
$$= 5x1 + 7x8$$
$$= 5 + 56$$
$$= 61_{10}$$

$$111101_2 = 1x2^0 + 0x2^1 + 1x2^2 + 1x2^3 + 1x2^4 + 1x2^5$$
$$= 1x1 + 0x2 + 1x4 + 1x8 + 1x16 + 1x32$$
$$= 1 + 0 + 4 + 8 + 16 + 32$$
$$= 61_{10}$$

Notice that if we take the base 8 number of 75 and convert each digit to base 2, we have the same number as when we converted the base 10 number to base 2. i.e.

Convert 7 to base 2

```
   3
2 | 7     R=1

   1
2 | 3     R=1

   0
2 | 1     R=1              111₂
```

Convert 5 to base 2

```
   2
2 | 5     R=1

   1
2 | 2     R=0

   0
2 | 1     R=1              101₂
```

Therefore 75_8 - 111101 which is the same pattern of 1's & 0's as we got from converting from base 10 to base 2. What this really says that it is easier to convert any base 10 number to base 8 by continuous division, and then convert each digit of the base 8 number to base 2.

Let's look at another example. Convert 183_{10} to base 8 & to base 2.

```
    22
8 | 183   R=7

    2
8 | 22    R=6          267(8)

    0
8 | 2     R=2
```

```
    91
2 | 183   R=1

    45
2 | 91    R=1

    22
2 | 45    R=1

    11
2 | 22    R=0          10110111(2)

     5
2 | 11    R=1

     2
2 | 5     R=1

     1
2 | 2     R=0

     0
2 | 1     R=1
```

$267_8 = 7x8^0 + 6x8^1 + 2x8^2$
$= 7x1 + 6x8 + 2x64$
$= 7 + 48 + 128$
$= 183$

to convert 267_8 directly to base 2, we convert each base 8 digit separately.

```
2      1
    2 | 2     R=1

       0
    2 | 1     R=1              10₂
```

```
6      3
    2 | 6     R=0

       1
    2 | 3     R=1              110₂

       0
    2 | 1     R=1
```

```
7      3
    2 | 7     R=1

       1
    2 | 3     R=1              111₂

       0
    2 | 1     R=1
```

therefore $2_8 = 10_2$, $6_8 = 110_2$, & $7_8 = 111_2$, and

$267_8 = 10110111_2$

Digital computers are designed to use binary numbers in their working registers. The working registers vary in number of bits depending on the manufacturer. The Motorola M6800 micro-processor utilizes, in general, 8 bit words (or registers). This leads to another number base, not yet mentioned, of hexadecimal. Hexadecimal is really a base 16 number system and can be handled in exactly the same manner as base 8 or base 2. In hexadecimal, four bits (in binary)represents one hexadecimal number. Thus , an eight bit register can be represented by a hex number of 2 digits long. To illustrate, lets assume we have the number of 147_8 in an eight bit register. This in binary form is 01100111 . If this bit pattern is divided into 2-four bit words of 0110 & 0111, then in hex, 147_8 can be represented as 67_{16}. To prove both are equal, lets convert both back to their base 10 number.

$147_8 = 7x8^0 + 4x8^1 + 1x8^2$
$= 7x1 + 4x8 + 1x64$
$= 7 + 32 + 64$
$= 103_{10}$

$67_{16} = 7x16^0 + 6x16^1$
$= 7x1 + 6x16$
$= 7 + 96$
$= 103_{10}$

As you probably have wondered by now, how do we represent these hex (base 16) numbers above 9? Here is the base 16 number compared with its equivalent base 10 number.

Base 10	Base 16
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

To convert any base 10 number to hex (base 16) you may convert it to base 8 first, then represent the base 8 number with its binary representation. By taking the binary representation of the number and grouping the bits from right to left in groups of four which are then represented in hex per the above table. Or one may convert any base 10 number to hex by our continuous division rule as before. Lets convert 825_{10} to hex.

```
     51
16 | 825   R=9

      3
16 | 51    R=3          339(16)

     0
16 | 3     R=3
```

therefore $825_{10} = 339_{16}$

to convert 339_{16} back to our base 10 number,

$339_{16} = 9x16^0 + 3x16^1 + 3x16^2$

$= 9x1 + 3x16 + 3x256$

$= 9 + 48 + 768$

$= 825_{10}$

To show the relationship between hex, binary, and octal, lets convert 825_{10} to octal & to binary and then back to hex.

825_{10} to octal

```
   103
8 | 825   R=1

    12
8 | 103   R=7
    1
8 | 12    R=4
    0
8 | 1     R=1
```

1471_8

825_{10} to binary

```
   412
2 | 825   R=1
   206
2 | 412   R=0
   103
2 | 206   R=0
    51
2 | 103   R=1
   255
2 | 51    R=1
   12
2 | 25    R=1
    6
2 | 12    R=0
   3
2 | 6     R=0
   1
2 | 3     R=1
   0
2 | 1     R=1
```

1100111001_2

$825_{10} = 1471_8$

$= 1x8^0 + 7x8^1 + 4x8^2 + 1x8^3$

$= 1x1 + 7x8 + 4x64 + 1x512$

$= 1 + 56 + 256 + 512$

$= 825_{10}$

$825_{10} = 1100111001_2$
$= 1x2^0 + Ox2^1 + 0x2^2 + lx2^3 + 1x2^4 + 1x2^5 + 0x2^6 + 0x2^7 + lx2^8 + 1x2^9$
$= lxl + 0x2 + 0x4 + lx8 + 1x16 + lx32 + 0x64 + 0x128 + 1x256 + 1x512$
$= 1 + 0 + 0 + 8 + 16 + 32 + 0 + 0 + 256 + 512$
$= 825_{10}$

Or taking $147l_8$ and representing each digit by its binary representation, we get 1=001, 4=100, 7=111 & 1 = 001 which when put together equal 001100111001. Notice this is the same bit pattern as when we converted from base 10 to base 2. Now if we group this into three groups of four bits and then convert each group to its hex counterpart, we will have the number of 825_{10} represented in hex. 001100111001 = 0011 0011 1001 = 339_{16}. Notice this agrees with the result when we converted directly to hex from one base 10 number.

In summary, lets take the situation when an MPU 6800 8 bit register contains all 1's.

$11111111 = 1x2^0 + lx2^1 + lx2^2 + 1x2^3 + 1x2^4 + 1x2^5 + 1x2^6 + 1x2^7$
$= 1x1 + 1x2 + 1x4 + 1x8 + 1x16 + 1x32 + 1x64 + 1x128$
$= 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128$
$= 255_{10}$ or

$11111111 = 11\ 111\ 111$
$= 3\ \ 7\ \ 7 = 7x8^0 + 7x8^1 + 7x8^2$
$= 7x1 + 7x8 + 3x64$
$= 7 + 56 + 192$
$= 255_{10}$ or

$11111111 = 1111\ 1111$
$= F\ \ F_{16}$
$= 15x16^0 + 15x16^1$
$= 15x1 + 15x16$
$= 15 + 240$
$= 255_{10}$

Conversion Chart

Decimal	Octal	Hexadecimal	Binary
0	0	0	0000 0000
1	1	1	0000 0001
2	2	2	0000 0010
3	3	3	0000 0011
4	4	4	0000 0100
5	5	5	0000 0101
6	6	6	0000 0110
7	7	7	0000 0111
8	10	8	0000 1000
9	11	9	0000 1001
10	12	A	0000 1010
11	13	B	0000 1011
12	14	C	0000 1100
13	15	D	0000 1101
14	16	E	0000 1110
15	17	F	0000 1111
16	20	10	0001 0000
17	21	11	0001 0001
18	22	12	0001 0010
19	23	13	0001 0011
20	24	14	0001 0100
21	25	15	0001 0101
22	26	16	0001 0110
23	27	17	0001 0111
24	30	18	0001 1000
25	31	19	0001 1001
26	32	lA	0001 1010
27	33	1B	0001 1011
28	34	1C	0001 1100
29	35	1D	0001 1101
30	36	lE	0001 1110
31	37	1F	0001 1111
32	40	20	0010 0000
33	41	21	0010 0001
34	42	22	0010 0010
35	43	23	0010 0011
36	44	24	0010 0100
37	45	25	0010 0101
38	46	26	0010 0110
39	47	27	0010 0111
40	50	28	0010 1000